ELSEVIER

Journal of Computational and Applied Mathematics 82 (1997) 351–366

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Graphical user interfaces for numerical libraries

Venkat V.S.S. Sastry*

Department of Applied Mathematics & Operational Research, Cranfield University, RMCS, Shrivenham, SN6 8LA, UK

Received 2 September 1996; received in revised form 13 April 1997

Abstract

This paper presents a portable and inexpensive way of developing high-quality grapical user interface front-end components to high quality numerical libraries. The GUI components are developed using the public domain software Tool Command Language (Tcl) and the associated Tool Kit(Tk).

Several GUI designs are illustrated using the numerical libraries NAG FORTRAN77 (Mark 16)/FORTRAN90 (Release 1). The approach is generic and can be easily adopted to user written programs.

The paper concludes with a discussion of merits of the present approach.

Keywords: Problem-solving environments; Tcl/Tk; GUI

AMS classification: 68T35

1. Introduction

With increasing acceptance of graphical user interfaces (GUI) in various disciplines of scientific computing, there is a far greater need to provide similar GUI facilities to high quality library software. In this paper we examine an attractive approach which offers substantial potential to develop portable and inexpensive GUIs.

Traditional development of GUIs involves edit–compile–run cycle; and are typically based on C-based X toolkits such as Xt, Xview, Motif, etc. This approach requires significant programming expertise and time. In contrast the Tool Command Language (Tcl) and its associated X toolkit (Tk) [14], offer greater flexibility in developing GUIs. The programming expertise required is no more demanding than Shell/C-shell programming; and the interface components can be developed interactively. For an excellent tutorial introduction to Tcl/Tk, the reader may consult [8, 15, 20].

*E-mail: sastry@rmcs.cran.ac.uk.

PII S0377-0427(97)00092-7

Matlab [9] together with MAGNum [7, 12] provides a convenient interface to develop effective graphical user interfaces. However, its scope is limited in coverage of numerical routines (MAGNum Toolbox provides interface to a subset of NAG Library), widgets and event bindings.

In the present paper, several GUIs are developed to assess the feasibility of Tcl/Tk as a GUI development tool. These GUIs are tentatively grouped into three broad classes – (1) browser class, (2) problem class, (3) pipeline class. In browser class, the GUI provides access to complete collection of example programs to the library software. Note however, a typical GUI for a problem solving scenario will require components from each class. In problem class, the GUI provides access to one or more routines from the library. In pipeline class, the GUI enables the user to build a block diagram of collection of routines, where the output from one routine is passed as input to another as required. Each routine is represented as an icon with input and output ports.

As an example of browser class, **nagexf77** and **nagexf90** are described briefly in Section 2. Two designs **datafit** and **gui_d01** are described in Section 3 which provide an interface to a collection of routines from NAG library. The former interface illustrates extensive graphics interaction associated with one-dimensional data fitting routines; while the latter design illustrates a possible approach to deal with non-numeric user input such as defining integrand in quadrature routines. A simple visual programming environment is described briefly in Section 4, which is limited to dozen routines from the library. Finally, the suitability of Tcl/Tk as a GUI development tool is summarized in Section 5.

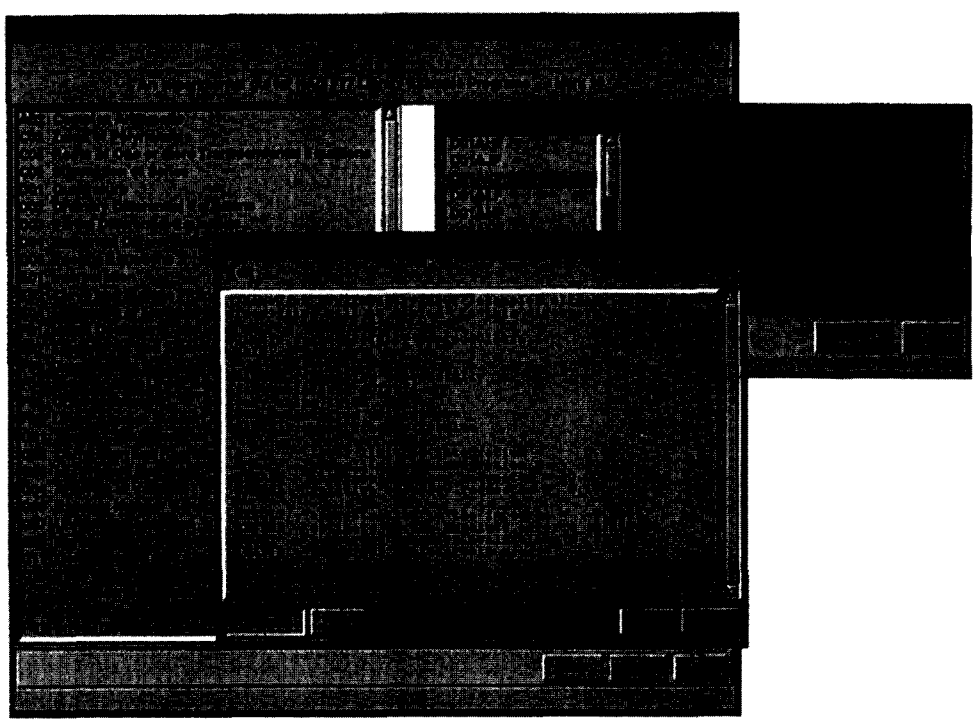

Fig. 1. **nagexf77** tool provides access to complete collection of example programs for FORTRAN77 library.

2. Browser Class – nagexf77 and nagexf90

There have been various attempts (e.g., [2, 16]) to provide help on using the library software. These earlier prototypes are limited to a subset of the library and as such are of limited practical use. Also, maintaining the GUI software with the new versions of the library software is a nontrivial task. Both these problems are addressed in this paper, by developing the required GUI components based on additional documentation files supplied with the library software.

As an illustration of this approach, two separate interfaces have been developed which provide access to complete collection of example programs supplied with NAG FORTRAN77 (Mark 16) and NAG FORTRAN90 (Release 1) libraries (see Figs. 1 and 2). The user can browse through the available Chapters and routines therein, copy and modify the source for example programs along with any required input data files, compile and execute the desired program. All the required textual information such as Chapter Titles, brief description of routines etc is derived from the vendor supplied text files which accompany the software distribution. This step keeps the maintenance of the GUI code to an absolute minimum.

In terms of functionality, both interfaces share several features. These are described below highlighting the differences where appropriate.

Selection of example programs. In **nagexf77** an example program is normally associated with a routine; while example programs are associated with a module name in **nagexf90**. Also in FORTRAN90 library, more than one example is provided for a given procedure/module. Once a desired procedure is selected, all the available example programs are listed for user's perusal.

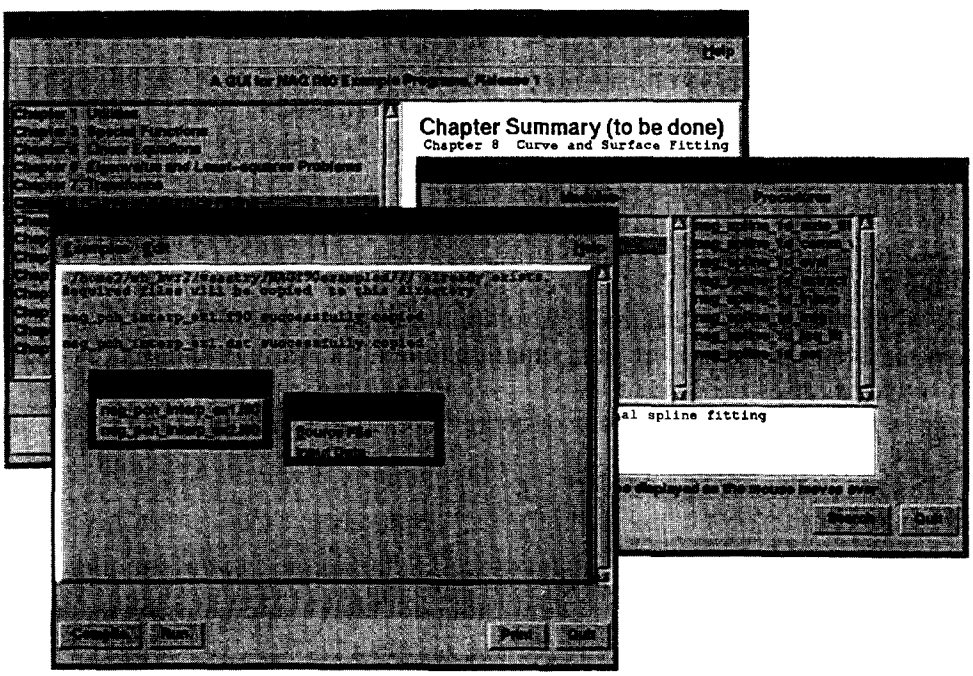

Fig. 2. **nagexf90** tool provides access to complete collection of example prorams for FORTRAN90 libraries.

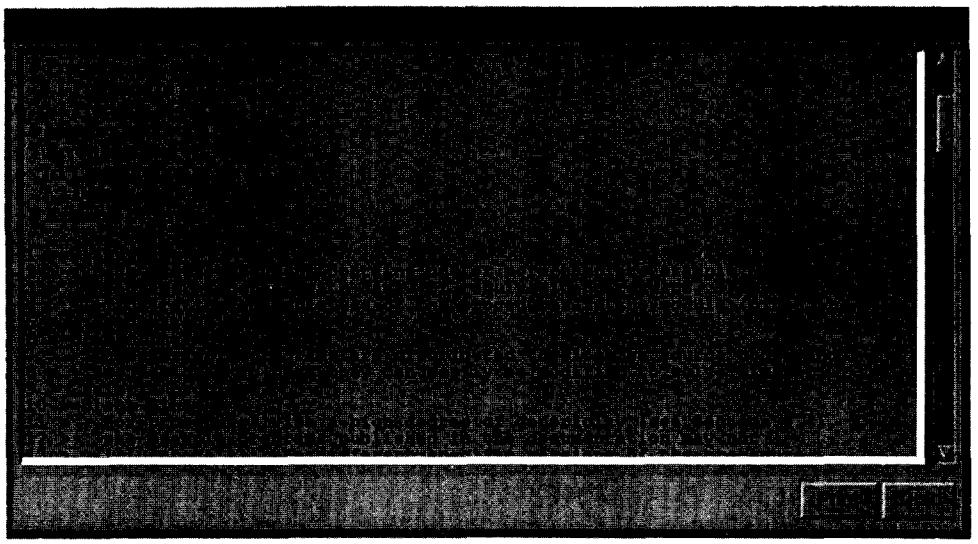

Fig. 3. Essential Introduction to FORTRAN90 library derived from simple text files.

Essential introduction Both libraries are supplied with simple text files (without any specific mark-up) containing useful information for novice as well as advanced user. Exploiting the partial mark-up, simple GUIs are developed to present this information in a hypertext format (see Fig. 3). Robust regular expressions are defined to identify the Section headings which follow a regular pattern.

A similar file containing Tutorial Introduction is also supplied with FORTRAN 90 library, and a similar GUI in hypertext format is currently under development.

Search facilities A simple search mechanism is provided at Chapter level. Summary file in each library contains a brief description of routine/procedure/module. A partial listing of these summary files is shown in Fig. 4.

Related information Additional interfaces to replaced/withdrawn routines and nag_to_blas_lapack names are provided in the case of nagexf 77. Similar interfaces are anticipated for FORTRAN 90 libraries as the required information becomes available.

At present the above interfaces are kept separate due to the intrinsic differences in the organization of respective libraries.

Table 1 gives an indication of amount of code involved in developing the above interfaces.

2.1. *Brief description of* **nagexf77**

In this section, development of the **nagexf77** interface is presented in some detail highlighting some of the interaction facilities. Complete listing of Tcl/Tk source program for **nagexf77** is available in [19]. For additional reference to Tcl/Tk, the reader is referred to [14, 20].

NAG FORTRAN 77 library is divided into several Chapters consisting of several routines. Most of the routines are supplied with a simple example program, an input data file (where appropriate)

```
D01AHF 1-D quadrature, adaptive, finite interval, strategy due to Patterson,
   suitable for well-behaved integrands
 D01AJF 1-D quadrature, adaptive, finite interval, strategy due to Piessens and
   de Doncker, allowing for badly-behaved integrands
 D01AKF 1-D quadrature, adaptive, finite interval, method suitable for
   oscillating functions


Chapter 11 === Quadrature

11.1 Module nag_quad_1d === Numerical integration over a finite interval
nag_quad_1d_gen === 1-d quadrature, adaptive, finite interval, allowing for
badly behaved integrand, allowing for singularities at user-specified
break-points, suitable for oscillatory integrands
nag_quad_1d_wt_trig === 1-d quadrature, adaptive, finite interval, weight
function cos(omega x) or sin(omega x)
```

Fig. 4. Typical contents of simple text files supplied with software distribution. They contain brief description of routines, modules and procedures. Partial mark-up in these files together with regular patterns are exploited in building hypertext type interface to these files. Simple search facilities are confined to the description of each routine, module or procedure as given in these files.

Table 1
Number of lines of Tcl/Tk code in **nagexf77** (left) and **nagexf90**

Program	No. of lines
NAGintro.tcl	644
essint.tcl	158
replaced.tcl	81
blas.tcl	102
search.tcl	102
naginit.tcl	25
Total	1112

File	No. of Lines
chap.tcl	153
epe.tcl	165
essint.tcl	222
f90ex.tcl	157
lunique.tcl	15
naginit.tcl	26
procs.tcl	423
search.tcl	140
Total	1301

and a sample output file. Some useful associated documentation is also supplied along with the software library. The source for the example programs and associated documentation (see Fig. 5) is made available to the end user.

Following the organization of the library, the interface is divided into three major components – an initial window (see Fig. 6) to present all the available Chapters and access to auxiliary support documentation; a chapter level window (see Fig. 7) to display all the available routines; and routine level window (see Fig. 8) to interact with a chosen example program.

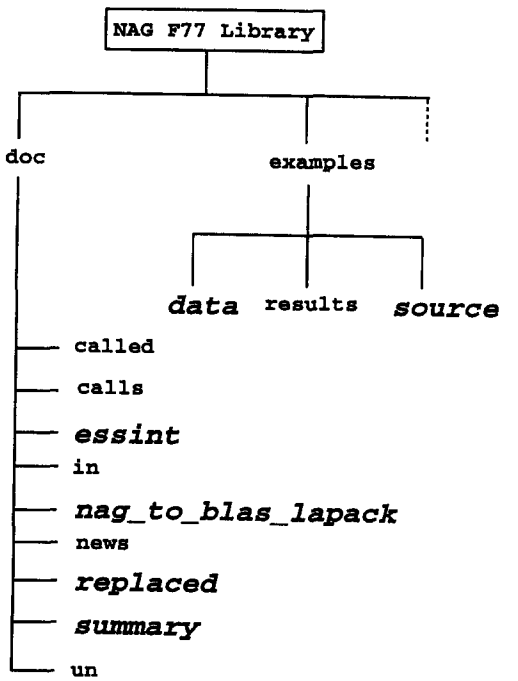

Fig. 5. Documentation supplied with NAG software. Text files shown in italics are used in building the interface.

The routine names in a Chapter follow a well defined pattern [3]. For example, a routine from the Chapter entitled D01 Quadrature is of the form D01XYZ where the Chapter code is followed by three alphabetical characters. The summary (cf. Fig. 4) file supplied with the library distribution contains the required information to obtain all the available routines in a chapter. Routine names and associated text describing their purpose is stored in an associative array for further processing.

The routine level window provides a simple environment to experiment (see Fig. 8) with the example program. Once the routine has been selected, the corresponding example program and any required input data file are copied into the user's area. The user may then edit, compile and run the program as required, and all the output from the program is placed in a text widget for user's perusal.

3. Problem class

This class of interaction is motivated from a desire to provide a simple interface specific to an application domain which require a collection of library routines. For example, in the area of one-dimensional data fitting, the user may wish to explore the quality of fit by inquiring derivative information at various points of the fitted curve. By grouping all the relevant routines together, the user is shielded from the precise details of invocation of these routines. Mandatory inputs to the

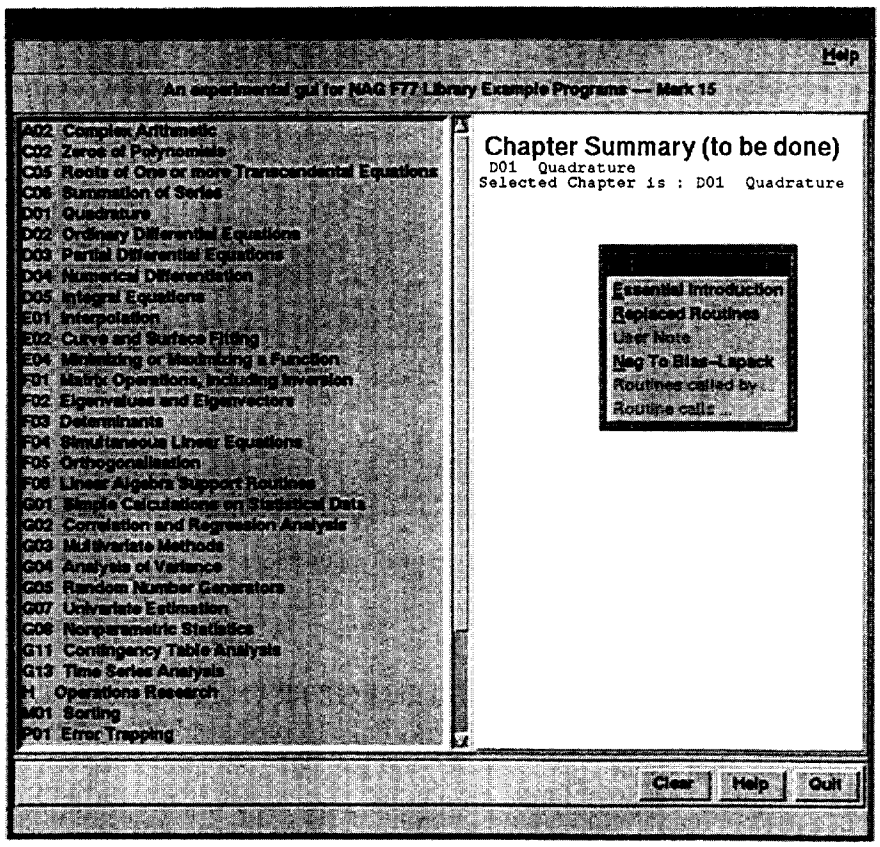

Fig. 6. Initial window of **nagexf77** displaying all the available chapters.

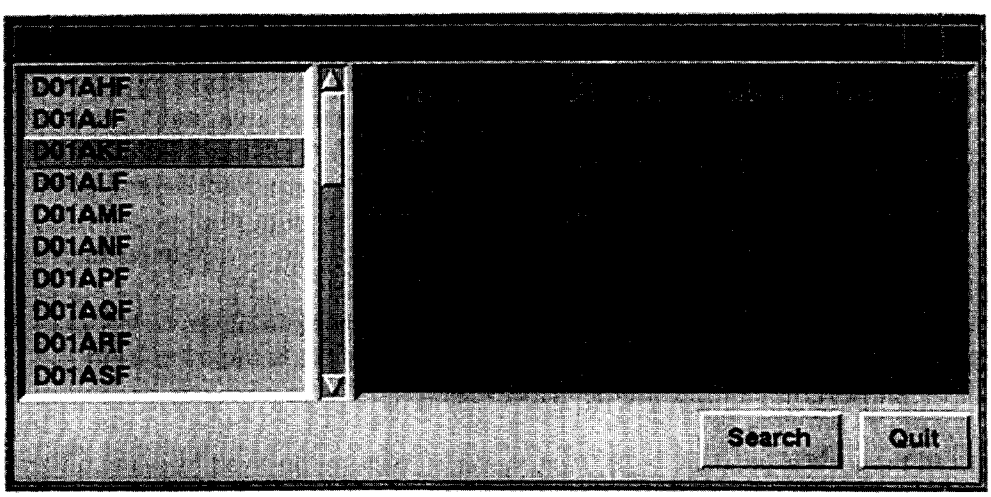

Fig. 7. A typical Chapter level window in which the left mouse button has been pressed on the routine D01AKF.

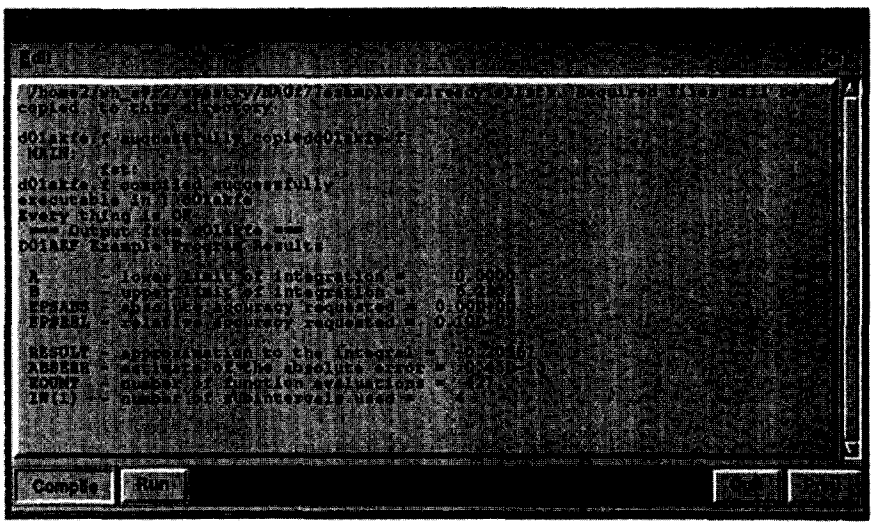

Fig. 8. A typical routine level window.

problem are collected from the GUI and passed to the FORTRAN executable(s). Depending on the application domain, required FORTRAN programs have to be developed in a consistent manner.

In this Section, two example interfaces – **datafit** and **gui_d01** – are presented which illustrate this class of interaction. Input to **datafit** application consists of numerical data and offers plenty of scope to illustrate graphics interaction, while the latter application **gui_d01** demonstrates the use of function definitions in input to the problem.

datafit In this example, an interface is developed for a collection of routines dealing with one-dimensional data fitting, where the user input consists of handling numerical data (via user's files or collected with mouse) and interrogation of the fitted curve.

The problem domain defines the choice of routines. For example, fitting splines to user supplied data, the user may wish to perform related computations such as area under the fitted curve and derivative information at a point on the fitted curve. Such a GUI has been built [13] using the NAG routines E01BAF (determines a cubic B-spline interpolant), E02BBF (evaluates the cubic spline at a point x), E02BCF (evaluates at the point x a cubic spline and its first three derivatives) and E02BDF (evaluates the definite integral of a cubic spline). The computed splines are displayed graphically (see Fig. 9) for further interaction.

In this application, each of the above routines have a corresponding driver program which read input from the standard input; and the output is written to the standard output.

Some of the important features are described below:

Invoking FORTRAN Executables FORTRAN executables are invoked by using unix pipes. Once, the required input is collected from the GUI, data is written to the pipe, and the program results are read from the pipe. Appropriate GUI components are updated as necessary. Alternatively, the

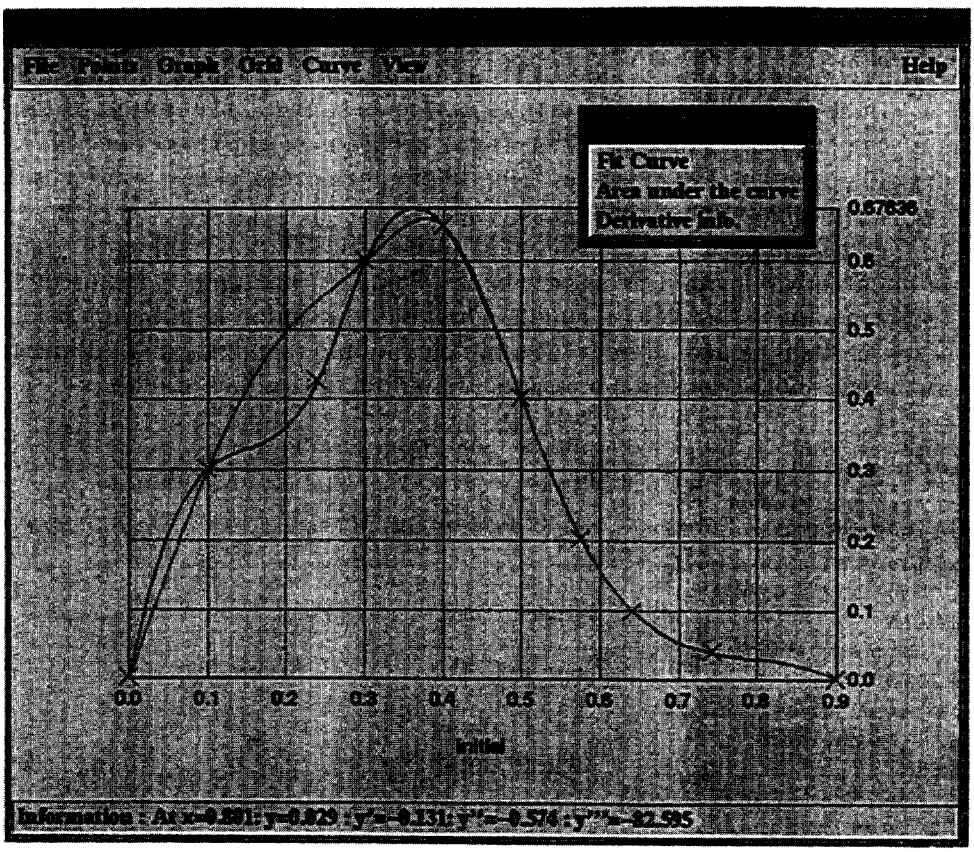

Fig. 9. datafit interface provides access to NAG routines E01BAF (determines a cubic B-spline interpolant), E02BBF (evaluates the cubic spline at a point x), E02BCF (evaluates at the point x a cubic spline and its first three derivatives) and E02BDF (evaluates the definite integral of a cubic spline).

FORTRAN executables are invoked by using redirection of standard input and standard output. A generic form of this mechanism in Tcl is

exec *ftn-exec*⟨input.dat⟩output.res

Required executables are prepared prior to invoking the interface. Output from these executables is used to update relevant interface components.

Graphics All the plotting routines are based on the graphics utilities provided in Tcl/Tk. Note that it is possible to exploit Tcl/Tk extensions such as BLT-wish [4] to further enhance graphics output.

Miscellaneous The interface provides several facilities to store/load data in/from a file, interact with data points, optionally display current and previous fit, display x and y labels, zoom in and out, etc.

gui_d01 In all the examples considered so far, the computational problems required simple alpha-numeric input data. In this example, a GUI is provided (see Fig. 11) to illustrate how one may handle function definitions required in several routines. A typical example is numerical quadrature, in which the integrand is normally supplied as a function subprogram. One approach is to define a template file from which the source program is generated once all the required information is gathered from the interfaces. Mandatory arguments to each routine is described in a separate file (see Fig. 10), from which the relevant GUI (Fig. 11) components are developed.

Each routine is associated with a *template* file in which mandatory arguments are marked. For example, an integrand is marked as shown below (line 15). The user is required to input the integrand following the FORTRAN syntax. Once all the input is entered, the appropriate FORTRAN source file is generated from the template file, and then compiled. All the output from the program together with diagnostic messages are written into a text widget.

```
1       DOUBLE PRECISION FUNCTION FST(X)
2  *     .. Scalar Arguments ..
3       DOUBLE PRECISION              X
4        double precision s17aef, s17aff
5  *     .. Scalars in Common ..
6       INTEGER                       KOUNT
7  *     .. Intrinsic Functions ..
8       INTRINSIC                     COS, SIN
9         external s17aef, s17aff
10 *     .. Common blocks ..
11      COMMON                        /TELNUM/KOUNT
12 *     .. Executable Statements ..
13      KOUNT = KOUNT + 1
14 c      FST = X*(SIN(30.0D0*X))*COS(20.08*X)
15      FST = <d01akf_fun>
16      RETURN
17      END
```

Note that the present interface is developed only to illustrate some of the user interaction that can be achieved in Tcl/Tk. Additional modifications will be required, for example to deal with integrands involving complex expressions (e.g., conditional expressions, handling of special functions in the integrand, etc).

4. Pipeline class – vpe

This model of interaction is largely inspired by the data-flow paradigm which has been extensively used in the development of several visualization systems (e.g., ([1, 5]). The principal idea is to represent each routine as an icon with input and output nodes. By selecting a collection of icons and connecting appropriate input/output nodes, the user develops a *data-flow map* to solve a computational problem. In this paper, these icons are also referred to as computational modules or simply as modules.

```
# { <rtn> {<input-1> {<comment-1>} {<default-1>} } ... }

set KB(d01ahf) { {a {lower limit}} {b {upper limit}}
                 {epsr {relative accuracy required}}
                 {nlimit {maximum function evaluations} 0} }

set KB(d01ajf) { {a {lower limit}} {b {upper limit}}
                {epsabs {absolute accuracy required} 0.0}
                {epsrel {relative accuracy required} 1.0D-04} }

set KB(d01akf) { {a {lower limit}} {b {upper limit}}
                 {epsabs {absolute accuracy required} 0.0}
                 {epsrel {relative accuracy required} 1.0D-04} }

set KB(d01amf) { {a {lower limit} -Inf}
                 {b {upper limit} +Inf}
                 {inf {type of integration range} 2}
                 {epsabs {absolute accuracy required} 0.0}
                 {epsrel {relative accuracy required} 1.0D-04} }

set KB(d01anf) { {a {lower limit}} {b {upper limit}}
                 {omega {parameter in the weight function}}
                 {key {1 for cosine, 2 for sine}} }

...
```

Fig. 10. Partial listing of file describing mandatory arguments of quadrature routines.

vpe is a visual programming environment written in Tcl/Tk.[1] It enables the user to develop a *data-flow map* of his problem by connecting the icons from the simple building blocks available.

In the present prototype, the following assumptions are made in developing the GUI environment:

- Each icon/module corresponds to a unique FORTRAN program and an associated executable. FORTRAN programs are generated from an accompanying template file. A template file is a driver program for a library routine where the required inputs to the problem are specially marked for subsequent replacement.
- Inputs to each module are normally read from a file. Some data is held in *global memory* of the problem solving environment. Relevant data is passed to the other modules as required. Where possible, inputs to the nodes are discerned (e.g., length of a vector).
- An output node may be connected to more than one input nodes.
- Data-flow network is executed in a serial fashion. Each module is run in a sub-shell to facilitate independent execution of a module.

Knowledge about each computational module is described in a resource file. Each entry consists of an enumerated list of input and output variables, their types (e.g., real scalar, integer array, etc.), and any default values to the input variables. Information regarding any work arrays is derived

[1]Total number of lines is approximately 2400 excluding source code for the FORTRAN driver programs.

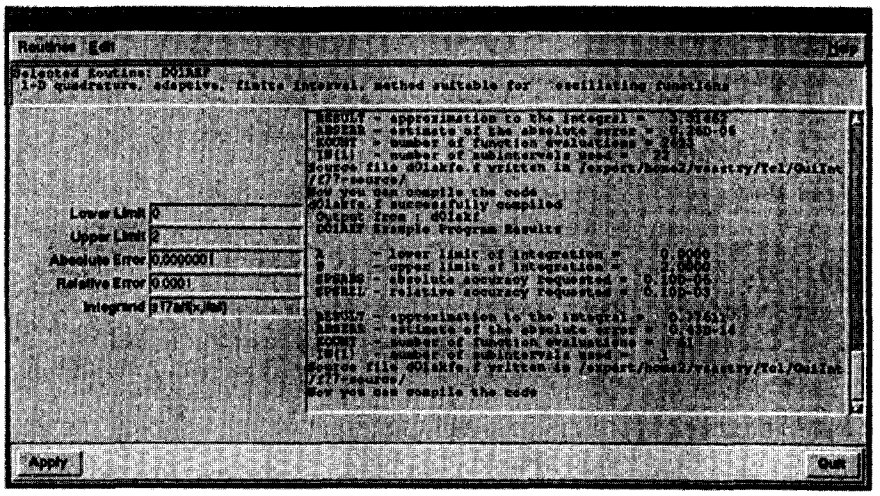

Fig. 11. A prototype interface for collection of routines dealing with numerical quadrature.

from the supplied information. The generic form of this resource file is

⟨icon_name⟩ [[⟨varname⟩ ⟨vartype⟩ ⟨vardef⟩]...]⟨output⟩

where ⟨icon_name⟩ is simply the name of the icon, and the triplet [⟨varname⟩ ⟨vartype⟩ ⟨vardef⟩] is repeated for each of the inputs to the icon, and contains the name, type and default value (if any) for each of the inputs. ⟨output⟩ is just a list of how many outputs the icon has, and whether they are scalars or arrays. For example, the NAG routine c02aef to find all the roots of a real polynomial equation is invoked as

SUBROUTINE C02AEF(A, N, REZ, IMZ, TOL, IFAIL)

where $A(N)$ are the coefficients of the polynomial. The computed roots are returned in real arrays REZ and IMZ. This information is represented as

```
nag_c02aef n i 10 a a:n { } "s a a"
```

This line tells *vpe* that the icon named "nag_c02aef" has two inputs – the first of which is an integer n, with a default value of 10, and the second is an array a, of size n, with no default value. Also there are three outputs from the module – the first of which is a scalar (containing the IFAIL parameter), and the remaining two are arrays (containing real and imaginary parts of the roots) which are required to be of size N.

nag_c02aef is associated with a template file in which the inputs to the problem—size parameter N and the array containing the coefficients are marked as shown below:

```
      program c02aef
*TCL_parameters
      integer          n
      parameter        (n = %TCL_n%)
      ...
```

```
      double precision a(n), imz(n), rez(n)
      external          c02aef
*TCL_arrays
      ...
```

The mark-up is replaced with appropriate information before compilation. Currently, this mark-up is used to pass both scalars, arrays, function statements and additional parameters local to the module.

As an example of dealing with function arguments, consider the routine d01akf which computes an approximation to a finite integral using an adaptive method, especially suited to oscillating, non-singular integrands. The subroutine is invoked as

```
SUBROUTINE D01AKF (F, A, B, EPSABS, EPSREL, RESULT, ABSERR, W, LW, IW,
                   LIW, IFAIL)
```

Knowledge about this routine is represented as

```
nag_d01akf a d {} b d {} epsabs d 1.0D-03 epsrel d 1.0D-03
           f(x) f {} "s s s"
```

where the mandatory inputs to the problem are lower and upper limit of integration (a and b) and the integrand (f(x)) (Fig. 12.) Other parameters, epsabs and epsrel to the subroutine are supplied with default values. The name of the function and its argument used in defining the integrand are described as a triplet as for other inputs to the problem. This module returns three scalar outputs, ifail parameter, approximation to the integral and an estimate of the modulus of the absolute error. Mark-up used for the integrand is shown below:

```
      double precision function f(x)
      double precision x
*TCl_parameters
      f = %TCL_f(x)%
      return
      end
```

Once the user has chosen what icons to use within *vpe*, and they are connected as required, each icon may be compiled and run individually, or the whole *network* may be compiled and run as a unit, and then display the outputs.

Currently, these icons have been developed for the following NAG subroutines:

a02aaf, a02abf, a02acf, c02aef, c02aff, d01ahf,
d01ajf, d01akf, f01brf, f04axf, f04jgf and g05faf

and user written program to set up the matrix required to invoke the NAG routine f04jgf (see Fig. 13).

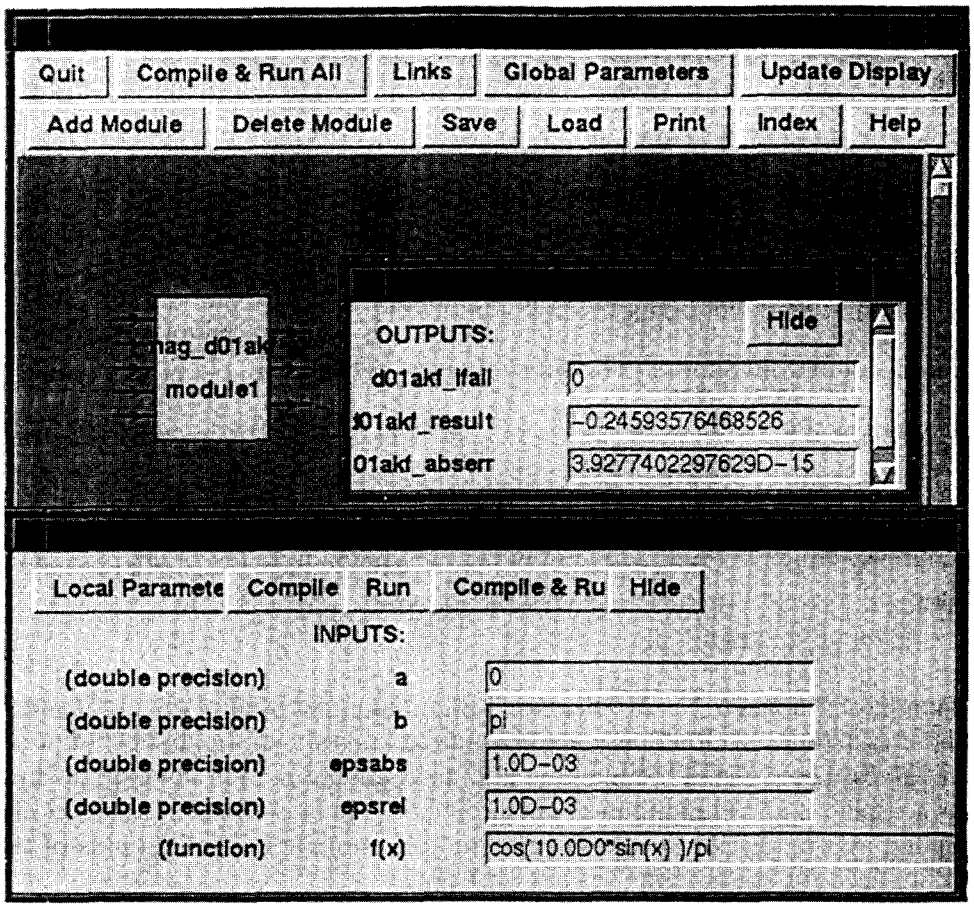

Fig. 12. Local parameters and definition of integrand. Approximation for $J_0(10) = \frac{1}{\pi}\int_0^{\pi} \cos(10\sin(x))\,dx$.

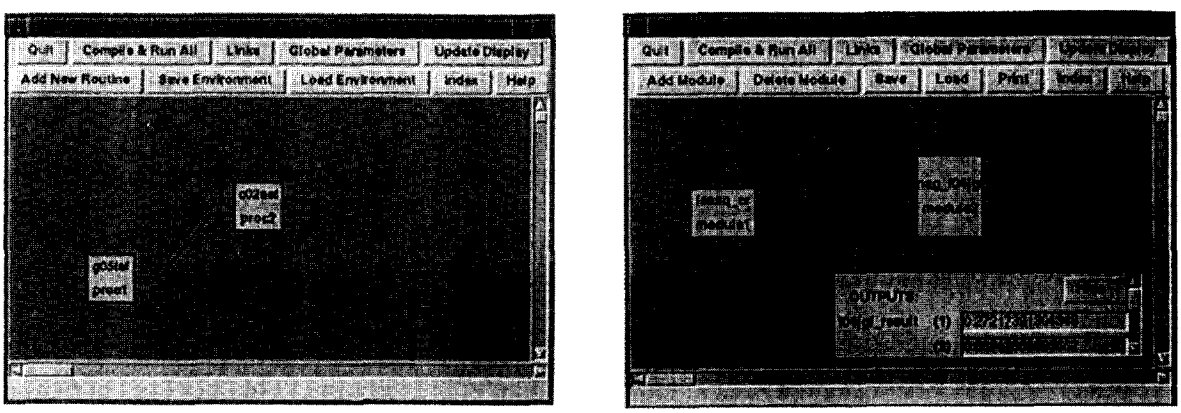

Fig. 13. A simple network to find the zeros of polynomial with random coefficients (left), and a network to assess least square fit to a set of data $\{x_i, y_i\}$.

Additional icons can be easily incorporated into *vpe*, by supplying a relevant template file, and a corresponding knowledge of the module in the resource file as described above. Template files can be easily generated by modifying the example programs with the software. However, great care is needed in developing these template files. It is our intention to provide support tools to create and maintain computational modules as an integral part of *vpe*. Once these support tools are available, *vpe* provides a convenient interface to build sophisticated problem solving *maps*, thus enhancing the usability of library software.

Currently template files are written in FORTRAN77, and there is scope to include template files written in other languages such as C, Fortran90, or Ada. There is also the possibility of adding template files written in matlab, *mathematica* or other packages languages by exploiting the Tcl extension, expect [6].

5. Summary

In this paper, the applicability of Tcl/Tk to build graphical user interfaces has been demonstrated by developing several GUI designs that provides easy access to high quality numerical library software such as NAG FORTRAN77 and FORTRAN90 libraries. The Tool Command Language provides a range of language features and utility functions ideally suited for the current task. Some of these facilities – string/list manipulation, file handling and communicating with Unix processes – have been fully exploited in the applications developed in this paper.

For the purposes of the present work, the GUI designs are classified into three classes – Browser, Problem and Pipeline class. A representative example from each class is presented together with a brief description of their functionality. For all the design developed in the present paper. Tcl/Tk has been found more than adequate in terms of functionality of the language, and the collection of available widgets. Note that Tcl/Tk is an extensible language, and offers the end user to define additional features as required.

No formal evaluation of the GUIs have been performed. However, all the GUIs presented in this paper have been used regularly in the class room, and the user's have reported substantial gains in turn-around-time for the results. The interface presented in Pipeline Class merits some further consideration. In particular, a thorough study comparing the present approach with that of developing similar interfaces using application builders (e.g., Iris Explorer or AVS) will highlight the merits and demerits of the current approach. Such a study is beyond the scope of the present paper.

On the whole Tcl/Tk has provided an excellent tool to develop various prototypes of the interfaces in fairly short period, and assess their suitability. The programming environment offered by Tcl/Tk is well within the reach of any seasoned programmer in any high level language; and provides a smooth migration path to advanced facilities such as creating new Tcl commands, creating new widgets etc should the need arise at a later date. The interactive nature of Tcl/Tk is an asset, particularly when developing working prototypes or proof-of-concept designs. Currently, the portability of these designs is limited to Unix workstations; and the author is awaiting more reliable ports of Tcl/Tk to Windows 3.1/NT. As a platform to develop efficient GUIs, Tcl/Tk offers much more than the author was able to explore in the present paper.

Acknowledgements

The author would like to acknowledge Richard Norgate for the code developed in Section 4 and **datafit** program. The author also wishes to thank the anonymous referees for their helpful criticism and suggestions. NAG is a registered trademark of Numerical Algorithms Group Limited . The programs developed in this paper can be obtained from the author and require Tcl/(Version 7.4)/Tk (Version 4.0), and relevant NAG FORTRAN libraries.

References

[1] Application Visualization System (Version 6), AVS Inc., 1996.
[2] C.W. Cryer, Expert Systems for Numerical Software, in: J.C. Mason, M.G. Cox (Eds.), Scientific Software Systems, Chapman & Hall, London, 1990.
[3] B. Ford, J. Bentley, A library design for all parties, in: D. Jacobs (Ed.), Numerical Software – Needs and Availability, Academic Press, New York, 1978, pp. 3–20.
[4] G. Howlett, M. McLennan, BLT Package (a Tk extension), ftp.aud.alcatel.com:tcl/extensions/BLT-1.9.0.tar.gz.
[5] Iris Explorer (Version 3.0), NAG Ltd., 1995.
[6] D. Libes, Exploring Expect, O'Reilly & Associates, 1995.
[7] MAGNum Handbook, NAG Ltd., 1993.
[8] G.A. Mark, A tutorial introduction to Tcl and Tk, The X Resource, Issue 11, 1995.
[9] Matlab, The Math Works Inc., 1992.
[10] NAG fl90 Manual, Vol. 1 and 2, (Release 1), NAG Ltd., 1994.
[11] NAG Fortran Library Manual (Mark 16), NAG Ltd., 1993.
[12] NAGWare Gateway Generator, NAG Ltd., 2nd Ed., 1994.
[13] R.E.J. Norgate, A visual programming environment for mathematical modelling, M.Sc. Project Report, 1995.
[14] J.K. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley, Reading, MA, 1994.
[15] L. Sastry, V.V.S.S. Sastry, Tcl/Tk Cookbook, http://www.cis.rl.ac.uk/proj/Tcl/Tk, 1996.
[16] V.V.S.S. Sastry, J.C. Mason, Knowledge based front-end to NAG library – KASTLE, Math. Comput. Simul. 36 (1994) 281–292.
[17] V.V.S.S. Sastry, NAGexTool – A GUI Front-end to NAG FORTRAN77 Library Example Programs, AMOR 95/8, 1995.
[18] V.V.S.S. Sastry, NAGex Tool – Reference Manual, AMOR 95/9, 1995.
[19] V.V.S.S. Sastry, nagexf77, ftp://www.rmcs.cran.ac.uk/pub/maths/tcltk/nagex77.tar.Z.
[20] B. Welch, Practical Programming in Tcl and Tk, Prentice-Hall, Englewood Cliffs, NJ, 1995.